Length Adjustment **835** for the Microscopy Device **805** to adjust any focus issues. This Machine Learning Model **833** trained by monitoring, over time, how the Microscopy Device **805** is adjusted in response to the acquired microscopy images and the output of the Trained CNN **830**. Such monitoring may be performed, for example, by recording instructions sent to the Microscopy Device **805**. Alternatively, an operator can manually enter the focal length changes into the Image Processing System **850**. Using the monitored data, a manifold (i.e., a basis set) of well-focused images can be learned that provides the correlation between the focal length and the quality of the image. Example techniques that can be employed to learn the manifold include, without limitation, principal component analysis (PCA), locally-linear embedding, and diffusion maps.

[0036] The Machine Learning Model **833** outputs a Focal Length Adjustment **835** for the Microscopy Device **805**. This Focal Length Adjustment **835** is then used as input to an Instruction Generator **840** that translates the adjustment into Executable Instructions **845** for the Microscopy Device **805**. The implementation of the Instruction Generator **840** is dependent on the interface of the Microscopy Device **805**. However, in general, the Instruction Generator **840** can be understood as software that provides an additional interface layer between the Image Processing System **850** and the Microscopy Device **805**. In some embodiments, the Machine Learning Model **833** can be trained to directly output the Executable Instructions **845**, thus obviating the need for the Instruction Generator **840**.

[0037] FIG. **9** provides an example of a parallel processing memory architecture **900** that may be utilized by an image processing system, according to some embodiments of the present invention. This architecture **900** may be used in embodiments of the present invention where NVIDIA™ CUDA (or a similar parallel computing platform) is used. The architecture includes a host computing unit ("host") **905** and a GPU device ("device") **910** connected via a bus **915** (e.g., a PCIe bus). The host **905** includes the central processing unit, or "CPU" (not shown in FIG. **9**) and host memory **925** accessible to the CPU. The device **910** includes the graphics processing unit (GPU) and its associated memory **920**, referred to herein as device memory. The device memory **920** may include various types of memory, each optimized for different memory usages. For example, in some embodiments, the device memory includes global memory, constant memory, and texture memory.

[0038] Parallel portions of a CNN may be executed on the architecture **900** as "device kernels" or simply "kernels." A kernel comprises parameterized code configured to perform a particular function. The parallel computing platform is configured to execute these kernels in an optimal manner across the architecture **900** based on parameters, settings, and other selections provided by the user. Additionally, in some embodiments, the parallel computing platform may include additional functionality to allow for automatic processing of kernels in an optimal manner with minimal input provided by the user.

[0039] The processing required for each kernel is performed by grid of thread blocks (described in greater detail below). Using concurrent kernel execution, streams, and synchronization with lightweight events, the architecture **900** of FIG. **9** (or similar architectures) may be used to parallelize training of the CNN. For example, in some embodiments, processing of individual cell images may be performed in parallel.

[0040] The device **910** includes one or more thread blocks **930** which represent the computation unit of the device **910**. The term thread block refers to a group of threads that can cooperate via shared memory and synchronize their execution to coordinate memory accesses. For example, in FIG. **9**, threads **940**, **945** and **950** operate in thread block **930** and access shared memory **935**. Depending on the parallel computing platform used, thread blocks may be organized in a grid structure. A computation or series of computations may then be mapped onto this grid. For example, in embodiments utilizing CUDA, computations may be mapped on one-, two-, or three-dimensional grids. Each grid contains multiple thread blocks, and each thread block contains multiple threads. For example, in FIG. **9**, the thread blocks **930** are organized in a two dimensional grid structure with m+1 rows and n+1 columns. Generally, threads in different thread blocks of the same grid cannot communicate or synchronize with each other. However, thread blocks in the same grid can run on the same multiprocessor within the GPU at the same time. The number of threads in each thread block may be limited by hardware or software constraints. In some embodiments, processing of subsets of the training data or operations performed by the algorithms discussed herein may be partitioned over thread blocks automatically by the parallel computing platform software. However, in other embodiments, the individual thread blocks can be selected and configured to optimize training of the CNN. For example, in one embodiment, each thread block is assigned an individual cell image or group of related cell images.

[0041] Continuing with reference to FIG. **9**, registers **955**, **960**, and **965** represent the fast memory available to thread block **930**. Each register is only accessible by a single thread. Thus, for example, register **955** may only be accessed by thread **940**. Conversely, shared memory is allocated per thread block, so all threads in the block have access to the same shared memory. Thus, shared memory **935** is designed to be accessed, in parallel, by each thread **940**, **945**, and **950** in thread block **930**. Threads can access data in shared memory **935** loaded from device memory **920** by other threads within the same thread block (e.g., thread block **930**). The device memory **920** is accessed by all blocks of the grid and may be implemented using, for example, Dynamic Random-Access Memory (DRAM).

[0042] Each thread can have one or more levels of memory access. For example, in the architecture **900** of FIG. **9**, each thread may have three levels of memory access. First, each thread **940**, **945**, **950**, can read and write to its corresponding registers **955**, **960**, and **965**. Registers provide the fastest memory access to threads because there are no synchronization issues and the register is generally located close to a multiprocessor executing the thread. Second, each thread **940**, **945**, **950** in thread block **930**, may read and write data to the shared memory **935** corresponding to that block **930**. Generally, the time required for a thread to access shared memory exceeds that of register access due to the need to synchronize access among all the threads in the thread block. However, like the registers in the thread block, the shared memory is typically located close to the multiprocessor executing the threads. The third level of memory access allows all threads on the device **910** to read and/or write to the device memory. Device memory requires the